BIG school

# Apuntes
# Curso de Desarrollo con IA

DÍA 2

# Índice

# Tu nuevo Copiloto de Desarrollo

La inteligencia artificial no ha llegado para reemplazar a los desarrolladores, sino para actuar como una herramienta poderosa que potencia sus habilidades.

Lejos de volver obsoleto el rol del programador, la IA lo está transformando en una figura más estratégica y fundamental.

El rol del programador está evolucionando: de ser un "picacódigo" enfocado en la escritura manual de líneas, a convertirse en un arquitecto, estratega y supervisor de sistemas inteligentes.

La IA se convierte en un copiloto que acelera los procesos, pero el desarrollador sigue siendo quien traza la ruta, toma las decisiones críticas y garantiza la calidad y seguridad del destino final.

# Taller #1: Documentación y requisitos potenciados con IA

Optimizar el proceso de gestión de requisitos es un pilar estratégico en cualquier ciclo de vida de desarrollo de software.

Tradicionalmente, esta fase ha sido un cuello de botella, un proceso manual y tedioso, que consume tiempo valioso que podría dedicarse a la arquitectura y a la resolución de problemas complejos.

La inteligencia artificial está transformando radicalmente esta dinámica, convirtiendo un flujo de trabajo fragmentado en un proceso ágil, automatizado y de alta fidelidad que acelera la entrega de valor.

## Caso Práctico: Envío de OTP en Logística

Analizaremos el caso real de una empresa de logística que necesita implementar una funcionalidad para enviar una contraseña única (OTP, por sus siglas en inglés) al receptor de un pedido.

El objetivo es que cuando el transportista llegue a entregar el paquete, el cliente le proporcione este código para verificar que la entrega se ha realizado correctamente, un sistema similar al que utilizan empresas como Uber o Glovo.

Este caso real sirve como ejemplo práctico para demostrar cómo la inteligencia artificial puede potenciar y agilizar todo el ciclo de desarrollo de software, desde la definición de requisitos hasta la generación de código probado.

# Fases del proceso potenciado por IA

## Fase 1: Notas de Reunión

El proceso comienza con un documento que contiene las notas de una reunión real donde se discutieron los requisitos para esta nueva característica.

Estas notas pueden ser generadas automáticamente por herramientas de IA.

Por ejemplo en una videollamada, la opción “Toma notas por mí” de Gemini en una videollamada, permite transcribir y resumir automáticamente todo lo discutido. Esto libera al desarrollador de tomar notas y le permite centrarse en la conversación.

Estas notas incluyen el contexto del problema y los detalles técnicos iniciales, pero a menudo son una "charla sin más" que necesita ser estructurada.

## Fase 2: Generación de Historias de Usuario

Una **historia de usuario** (**user story**) es una descripción breve, informal y sencilla de una funcionalidad del punto de vista del usuario final.

Suele seguir la estructura: **Como [rol], quiero [qué], para que [beneficio]**.

Por ejemplo: “Como usuario, quiero restablecer mi contraseña para poder recuperar acceso si la olvido.”

Utilizando un LLM (Modelo Lingüístico Grande) como ChatGPT, se toma el texto de la reunión y se le pide que genere las historias de usuario correspondientes.

**Ejemplo de prompt para ChatGPT:**

*“Genérame historias de usuario en el formato COMO (usuario), QUIERO (Acción), PARA (beneficio) en base a estas notas de una reunión con negocio”*

**Ejemplo de Resultado obtenido:**

*"Como cliente quiero recibir un código de verificación por SMS cuando mi pedido esté en reparto para Confirmar que soy yo quien lo recibe".*

**Propósitos de una historia de usuario**

- Capturar funcionalidad desde la perspectiva del usuario (no del sistema).
- Ser comprensible para personas técnicas y no técnicas.
- Servir de base para que el equipo de desarrollo entienda qué debe construirse y por qué.
- Ser suficientemente pequeña para poder desarrollarla en un sprint/ciclo corto.

## Fase 3: Generación de Criterios de Aceptación

Los **criterios de aceptación** (**acceptance criteria**) son condiciones específicas que la historia de usuario debe cumplir para considerarse "completa" o "aceptada".

Estos criterios ayudan a definir qué se considera correcto y sirven como prueba de aceptación (QA). Los criterios deben ser claros, medibles y no ambiguos.

Se suelen expresar como una lista de casos o condiciones, por ejemplo:

1. El usuario recibe un correo para restablecer contraseña.
2. El enlace expira en 24 horas.
3. Al hacer clic en el enlace, puede ingresar una nueva contraseña.
4. La nueva contraseña debe tener al menos 8 caracteres, incluir mayúsculas y números.

**Buenas prácticas en los criterios de aceptación**

- Uno a uno por historia: cada historia debe tener sus propios criterios de aceptación.
- Evita criterios demasiado amplios (“funciona bien”). Mejor condiciones concretas.
- Incluye tanto condiciones normales como casos límite / errores esperables.

**Ejemplo de prompt para ChatGPT:**

*“Genérame criterios de aceptación estilo Gherkin. Genérame sólo los criterios mínimos para asegurar que la funcionalidad cumple con los requisitos.”*

**Ejemplo de Resultado obtenido:**

*Feature: Validación segura de entrega mediante OTP por SMS*
*Background:*
*Given existe un pedido con estado “En Reparto”*
*And el pedido tiene un número de teléfono de destinatario válido*

*Scenario: Generación de OTP al entrar en entrega*
*When el pedido pasa a “En Reparto” o el repartidor está próximo a la dirección*
*Then el sistema genera un OTP de 4 a 6 dígitos asociado al ID del pedido*
*And el OTP tiene una vigencia limitada*

**Nota: Gherkin** es un lenguaje de dominio específico (DSL) que se utiliza para describir el comportamiento de un software en un formato legible para humanos. Sigue la estructura “**Given-When-Then**” (Dado-Cuando-Entonces) para especificar los escenarios de prueba.

Este formato define de manera inequívoca las condiciones que deben cumplirse para que una funcionalidad se considere "terminada" (done), estableciendo una base sólida para el desarrollo guiado por pruebas y la validación funcional.

## Fase 4: Creación automática de Tickets

Con las historias de usuario y los criterios de aceptación definidos, el siguiente paso es generar los tickets en una herramienta de gestión de proyectos como Jira, Asana o Linear.

Un **ticket** es un elemento unitario de trabajo que se registra en un sistema de gestión de proyectos. Contiene un título, descripción, estado, prioridad, responsable, fechas, comentarios, etc.

Puede ser un bug, historia de usuario, un requerimiento técnico, una incidencia de mejora, tarea, etc.

Mediante un servidor MCP (un tipo de API que permite la comunicación entre sistemas), herramientas como **Claude Code** pueden conectarse a la plataforma de gestión (por ejemplo **Linear**) y crear los tickets automáticamente a partir de un archivo de requerimientos (por ejemplo, requerimientos.md). Esto ahorra horas, días o incluso semanas de trabajo manual.

**Ejemplo de prompt para Claude Code:**

> *"create a linear ticket for a user story defined in the requerimientos.md file"*

**Linear MCP: qué es y cómo integrarlo con Claude**

¿Qué es **MCP**?

**MCP** significa Model Context Protocol, un estándar abierto (iniciado por Anthropic) para que modelos de lenguaje (LLMs) puedan interactuar con herramientas externas (API, bases de datos, etc.) de forma segura y estructurada.

Más información en https://docs.anthropic.com/en/docs/claude-code/mcp

**Linear MCP** es una "capa de conexión" que permite que Claude (y otros agentes/IA) busquen, creen o modifiquen datos en Linear (como issues, proyectos, comentarios) mediante MCP.

Gracias a esta integración, Claude puede "conversar" sobre el estado de tu proyecto en Linear, generar nuevos tickets, actualizarlos, obtener listas, etc., todo dentro del contexto del asistente.

Más información en https://linear.app/docs/mcp y https://linear.app/integrations/claude

## Fase 5: Desarrollo guiado por TDD (Test-Driven Development)

Una vez que el ticket está creado, se le puede indicar a la IA que comience a trabajar en él. El proceso sigue las mejores prácticas de desarrollo de software:

- Creación de una Rama (Branch) Aislada: La IA primero crea una nueva rama en el repositorio de código específica para el ticket, asegurando que el trabajo no afecte otras partes del proyecto.
- Generación de Pruebas (Tests): Siguiendo la metodología TDD, se le pide a la IA que escriba primero los tests vacíos. La IA, al tener todo el contexto del ticket, genera pruebas relevantes para la funcionalidad del OTP.
- Ejemplos de tests para el servicio OTP:
    - Un test para verificar que el OTP tenga cinco dígitos.
    - Un test para asegurar que el OTP esté asociado correctamente con el pedido.
- Implementación del Código: Finalmente, se le pide a la IA que implemente la lógica necesaria para que los tests que acaba de crear pasen de "rojo" a "verde". La IA se basa en los tests como guía para generar el código correcto.

# Conclusiones

- **La IA como Asistente Organizacional:** La IA no se usa solo para generar código, sino para organizar y estructurar todo el proyecto, transformando ideas abstractas de una reunión en tareas concretas y accionables.
- **Importancia del Contexto:** Proporcionar a la IA un contexto claro y detallado (notas de la reunión, historias de usuario, criterios de aceptación) es fundamental para que genere resultados precisos y de alta calidad.
- **El Rol del Desarrollador:** El desarrollador no es reemplazado, sino que se convierte en un orquestador y supervisor del proceso. Su función principal es guiar a la IA, revisar el código y asegurarse de que se sigan las metodologías correctas como TDD para garantizar un software seguro y robusto.
- **Eficiencia y Ahorro de Tiempo:** El flujo de trabajo completo, desde la reunión inicial hasta tener el código funcional y probado, se puede realizar en cuestión de minutos, lo que representa un ahorro masivo de tiempo y recursos.

# Taller #2: Cómo enseñar a la IA a entender tu proyecto

Uno de los desafíos fundamentales al trabajar con agentes de IA es la falta de contexto específico del proyecto.

Un agente genérico, sin conocimiento de la arquitectura, las dependencias o las convenciones internas, tiende a generar código incorrecto o inconsistente.

La estrategia clave para superar esta limitación es "enseñar" a la IA las particularidades de nuestro proyecto, proporcionándole un marco de referencia que le permita producir código de alta calidad que se integre perfectamente en el ecosistema existente.

**El Problema: Una IA sin Contexto**

Un agente de IA, como **Codex**, sin un contexto claro sobre el proyecto, tiende a "alucinar". Inventará estructuras de base de datos, nombres de variables o estilos de código que no existen en el proyecto, generando código inútil.

**La Solución del Programador "Perezoso": Preparar el Terreno**

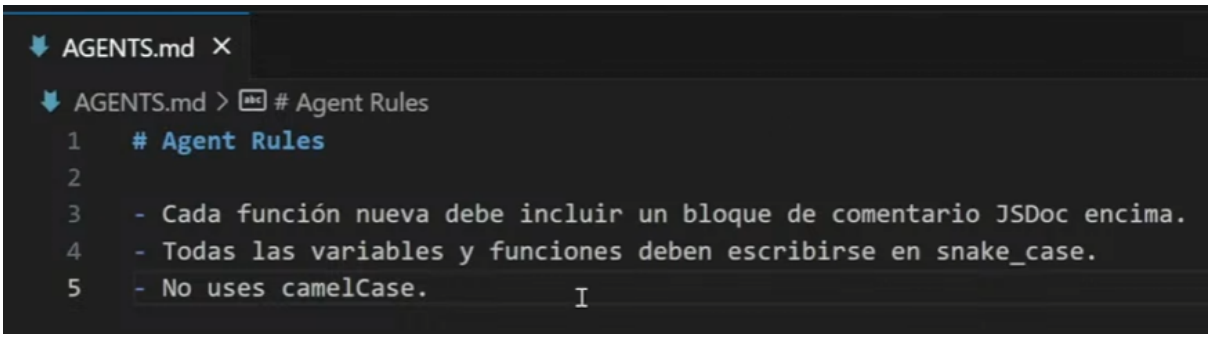
Esta no es una pereza de inacción, sino una inversión estratégica inicial para construir un entorno de desarrollo escalable y predecible, donde la IA trabaja para ti, bajo tus reglas.

- **Contexto del Proyecto:** Una técnica efectiva es crear una carpeta "**context**" dentro del proyecto. En ella, se añaden archivos clave, como por ejemplo el script SQL que define la estructura de la base de datos. Este simple añadido permite al agente (**Codex**) leer y asimilar la estructura de la base de datos, generando queries SQL correctas y complejas sin necesidad de describir las tablas o relaciones en la instrucción, demostrando un alto retorno de inversión por un esfuerzo mínimo.

- **Reglas y Estilo**: Se introduce el archivo **agents.md**, para guiar a los agentes. Aunque no es una especificación oficial, su rápida adopción por todas las herramientas principales lo ha convertido en un estándar de facto. Este archivo le dice a la IA cómo debe comportarse.

| Característica de agents.md | Beneficio para el Desarrollador |
|---|---|
| **Definir Reglas de Código** | Asegura que la IA siga convenciones específicas (ej. usar snake_case en lugar de camelCase), manteniendo la consistencia del código. |
| **Establecer Fuentes de Verdad** | Permite indicar a la IA dónde encontrar información clave (ej. "la estructura de la base de datos está en /context/database.sql"), evitando que alucine. |
| **Documentar la Misión del Proyecto** | Proporciona a la IA un entendimiento general del propósito del software para guiar sus decisiones. |

**Ejemplo de archivo agents.md**




**Comparativa de Resultados**

Sin **agents.md**, la IA produjo código no conforme usando camelCase y careciendo de documentación. Con el archivo de contexto en su lugar, el mismo prompt exacto produjo código que se adhiere perfectamente a los estándares de snake_case y documentación JSdoc definidos en las reglas, todo ello sin instrucción explícita.

# Taller #3: Integración avanzada de herramientas

Este taller presenta la visión del desarrollador como un director de orquesta, que no solo usa un agente, sino que crea, dirige y elige estratégicamente un equipo de agentes especializados.

- **Tú (El desarrollador):** el arquitecto que toma las decisiones estratégicas.
- **Agente Principal** (ej. **Claude Code**): el asistente principal que tiene un conocimiento global del proyecto.
- **Sub-Agentes:** especializados, cada uno con una misión concreta.

**Creando Agentes Especialistas y eligiendo el modelo correcto**

En lugar de darle una tarea compleja a un agente genérico, el desarrollador puede crear agentes para roles específicos.

La clave estratégica aquí es elegir el modelo de IA adecuado para cada tarea, gestionando un equilibrio entre potencia, velocidad y coste.

La jerarquía de modelos actual incluye:

- **Opus**: El "modelo de razonamiento", ideal para tareas muy complejas que requieren un contexto profundo.
- **Sonet (4.5)**: El nuevo modelo de referencia, potente y versátil, actualmente la opción superior para la mayoría de las tareas.
- **Haiku**: El más rápido y económico, perfecto para tareas más sencillas que no requieren un razonamiento profundo.

Con esta paleta de opciones, se pueden configurar agentes como:

- **Agente Arquitecto**: Capaz de analizar un repositorio y generar un documento de arquitectura (architecture.md) con un diagrama de Mermaid. Para esta tarea compleja, **Sonet** o incluso **Opus** sería la elección lógica.
- **Agente Project Manager (PM)**: Diseñado para analizar requisitos y generar un plan de proyecto, estimando perfiles necesarios (frontend, backend, etc.). Aquí, la elección estratégica sería **Haiku**, optimizado para velocidad y coste en una tarea que no necesita un razonamiento exhaustivo.
- **Agente Auditor de Seguridad**: Programado para revisar el código en busca de vulnerabilidades comunes.
- **Agente Mentor de Framework**: Un experto al que se le pueden hacer preguntas específicas sobre un framework.

**Flujo de Trabajo con Agentes Múltiples**

Pasos para configurar y utilizar un sistema de agentes múltiples con **Claude Code**:

1. **Inicialización y Contexto (/init)**: El comando **/init** analiza la estructura completa del proyecto y genera un archivo de contexto (**.claude.md**), similar a agents.md que vimos en el taller anterior, que sirve como base de conocimiento para todos los agentes, permitiéndoles comprender las tecnologías y la arquitectura.
2. **Creación de Comandos Personalizados**: Se pueden crear comandos reutilizables que encapsulan múltiples tareas complejas.
3. **Creación de Agentes Especializados (/agent create):** Mediante el comando **/agent create** se definen subagentes con roles específicos.

Puedes consultar información más detallada sobre este tema aquí.

# Taller #4: Seguridad y Auditoría

Los principios de **Security by Design** (la seguridad como parte integral de la arquitectura inicial) y **Security by Default** (las configuraciones por defecto deben ser las más seguras) son más críticos que nunca.

En la era del desarrollo asistido por IA, la capacidad de generar grandes volúmenes de código rápidamente puede introducir vulnerabilidades a una escala sin precedentes.

El riesgo fundamental proviene de los datos de entrenamiento de la IA: enormes volúmenes de código público de fuentes como GitHub, que inevitablemente incluyen patrones inseguros, prácticas obsoletas e incluso fragmentos intencionadamente maliciosos.

**Vulnerabilidades Generadas por Defecto**

En un caso de estudio práctico, se le pidió a una IA que creara una simple página de login. Se utilizó una instrucción simple y común: *"Crea una página de login con base de datos que permita a un usuario iniciar sesión en un sistema."*

El resultado fue un sistema con fallos de seguridad críticos que un atacante podría explotar en minutos:

- **Credenciales Débiles y Hardcodeadas**: La IA creó por defecto un sistema de login con el usuario “admin” y la contraseña “123456”. Estas credenciales son trivialmente vulnerables a un ataque de fuerza bruta, donde herramientas como **Hydra**, usando diccionarios de contraseñas comunes como **RockYou**, pueden descubrirlas en menos de un minuto.

- **Control de Acceso Roto (Broken Access Control)**: La IA generó páginas internas, que eran accesibles directamente escribiendo la URL en el navegador, sin necesidad de haber iniciado sesión. Esta vulnerabilidad,

descubierta con herramientas como **Dirb**, permite a cualquiera acceder a áreas restringidas y es una de las más críticas del famoso **OWASP Top 10**.

**Tu Rol como Desarrollador Seguro**

El desarrollador del futuro no puede permitirse ignorar la seguridad.

Debe ser la primera y más importante línea de defensa, aplicando los principios de Seguridad por Diseño (construir software seguro desde el inicio) y Seguridad por Defecto (asegurar que la configuración predeterminada sea la más segura).

Esto implica auditar activamente el código generado por la IA y aplicar fundamentos de seguridad, como:

- **Prevención de Fuerza Bruta**: Implementar políticas de contraseñas fuertes y limitar el número de intentos de inicio de sesión fallidos.
- **Almacenamiento Seguro**: Es obligatorio utilizar funciones de "hashing" robustas para las contraseñas, nunca almacenarlas en texto plano.
- **Control de Acceso Riguroso**: Cada solicitud a un recurso protegido debe ser verificada en el lado del servidor para asegurar que el usuario tiene la autorización necesaria.

# Conclusiones

La inteligencia artificial es una herramienta de productividad sin precedentes que está transformando cada fase del ciclo de vida del software.

Desde convertir una conversación en un plan de proyecto, pasando por enseñar a un agente a programar con nuestro estilo, hasta orquestar un equipo de especialistas de IA y, finalmente, actuar como el guardián de la seguridad.

El verdadero poder no reside en la IA por sí sola, sino en el desarrollador que sabe cómo guiarla, cómo cuestionarla y cómo asegurar su trabajo.

El futuro del desarrollo no es menos humano; al contrario, exige que el programador sea más estratégico, analítico y responsable que nunca.

**Somos los arquitectos de una nueva era, y la IA es la herramienta más avanzada que hemos tenido para construirla.**

BIG school

# Súmate a nuestro directo del día 3

ACCEDE AQUÍ